(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 507 552 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **92302833.6**

(22) Date of filing : **31.03.92**

(51) Int. Cl.[5] : **G11B 20/10**, G11B 20/18

(30) Priority : **02.04.91 US 679455**

(43) Date of publication of application : **07.10.92 Bulletin 92/41**

(84) Designated Contracting States : **DE FR GB**

(71) Applicant : **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor : **Menon, Jaishankar Moothedath**
6017 Montoro Drive
San Jose, CA 95120 (US)

(74) Representative : **Burt, Roger James, Dr.**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) Accessing variable length records stored on fixed block formatted DASDS.

(57) In order that variable length records be accessed from an array of (N+2) synchronous fixed block formatted DASDs in a single pass and in the presence of a single DASD failure, each record is partitioned into a variable number K of fixed length blocks, the blocks are written on the DASDs in column major order K modulo (N+1), the order is constrained such that the first block of each record resides on the (N+1)th DASD, a parity block for each column resides on an (N+2)th DASD, and each parity block spans N blocks in the same column from the first N DASDs and one block one column offset thereto on the (N+1)th DASD.

With four DASDs, DASD4 is reserved as parity DASD. In column major order, block 1 containing the home address HA is on DASD1, block 2 containing the record RO is on DASD2 and block 3 containing the count field C1 for record 1 is on DASD3. Blocks 4, 5 and 6, containing data for record 1 and the count field C2 for record 2 are on DASD2 1, 2 and 3, respectively. Blocks 7, 8 and 9, containing data for record 2, zeros and the count field for record 3 are similarly on DASDs 1, 2 and 3, respectively, and so on. All count fields are on DASD3. Parity from blocks 1 and 2 is stored in block P1 of DASD4. Parity from blocks 3, 4 and 5 is stored in block P2, and so on.

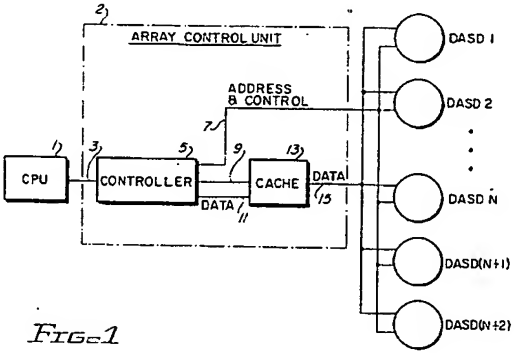

FIG. 1

EP 0 507 552 A2

This invention relates to accessing of variable length records to and from an array of N+2 synchronous fixed block formatted direct access storage devices (DASDs).

DASDs have been formatted to accommodate storing either variable length or fixed length records along their circular track extents. One persistent goal has been that of minimizing the time required to access such DASD stored records. In turn, this means minimizing the time taken to position radially a read/write head over a DASD track and track extent at the start of a record, and, minimizing the number of rotations required to stream records to or from the device. Always, it is necessary to readjust the system state (perform the necessary book-keeping) with each access.

US-A-4,223,390 describes track to track mapping of variable length records from a less dense to a more dense. DASD recording medium (also termed ONTO mapping). EP-A-0347032 discloses mapping variable length records as a virtual track overlay onto fixed block formatted tracks. A copending European patent application No. 92301133.2 discloses performing write updates of variable length records in row major order (DASD track direction) among elements of an N DASD array in a shortened interval.

A significant fraction of data reposes on DASD storage in variable length format. One regimen, which is used on IBM S/370 CPUs and attached external storage, is knowm as Count/Key/Data or CKD operating under the well known MVS operating system. In this regimen, each record consists of fixed length count field, an optional key field, and a variable length data field. The count field defines the length of the data field while the key field serves to identify the record.

The fields, as recorded on DASD tracks, are spaced apart along a track by gaps of predetermined fixed size. As the track rotates under a read/write head at a constant speed, such gap defines a time interval during which the system prepares to handle the next field.

A record having a data field spanning more than one physical DASD track extent is reformatted at the CPU into several smaller records with appropriate pointers or other linking conventions maintained in the storage or file management portions of the operating system. Likewise, a track may store many small records of various lengths. It follows that more processing is required to read, write, and update variable length records on DASD than records having only fixed extents.

In this specification, the terms "sector" and "track extent" are synonymous. When used in the context of am array of DASDs, the terms are synonymous with the term "column".

US-A-4,223,390 describes the mapping of variable length formatted (CKD) records from a full DASD track of lesser recording density onto a partial DASD track of greater recording density. An elaboration of counters and offsets is used in order to ensure one-to-one field, gap, and end of track correspondence and recording. This results from the difference in track lengths occupied by the same elements on the different density tracks. This is ONTO mapping in the sense that all of the elements of one set are a proper subset of the second set.

EP-A-0347032 relates to a method for partial track mapping of CKD variable length records onto fixed block formatted DASD tracks. The method steps include (a) blocking and recording of CKD records on the DASD using embedded pointers while preserving record field and gap order and recording extents, and (b) staging the blocked records defined between pointers in response to access commands.

The blocking step calls for (1) inserting a pointer at the beginning of each block to the first count field of any CKD record, if present. Gap information is encoded by the pointer to the count field as the start of record. Likewise, the count field points to counterpart key and data fields. The blocking step further calls for (2) inserting a pointer in the count field of the last CKD record indicating the end of the logical CKD track.

The last step of the method is responsive to CKD address information from a CPU generated read command. This step involves staging the block from the fixed block formatted DASD by way of accessing the block number counterpart to the CKD address and the path defined by the recorded pointers within said block until record end.

The copending European patent application No. 92301133.2 shows that write updating of variable length records stored in row major order (DASD track direction) on an array of N DASDs is facilitated by utilizing the correlation between byte offsets of a variable length record and the byte offset of a byte level parity image of data stored on the same track across N-1 other DASDs.

Thus, the write update in a shortened interval is obtained by altering and rewriting the parity concurent with altering and rewriting the data. That is, both data and relevant parity are accessed in terms of byte offsets in an equivalent virtual DASD. Then, the data and parity are recalculated and rewritten on the selected and Nth DASD respectively.

The parity images are distributed across different DASDs such that there is no "parity DASD" as such. For instance, for an array of N=1O DASDs, the image of the ith track from DASDs 1 to 9 would be stored on DASD 10 while the image of the ith+1 track over DASDs 2 to 10 would be stored on DASD 1.

Significantly, US-A-4,223,390 requires elaborate comparison counting of bytes and byte offsets of fields and gaps laid along a track in order to perform the CKD to CKD ONTO track mapping of records between

DASDs. This is caused by the dissimilarity of track recording densities.

While both EP-A-0347032 and the copending European patent application define variable length records over recording tracks having the same density, an elaboration of byte level counting, displacements, and pointers is still required. In EP-A-0347032, the elaboration is used to create the "virtual disk track of variable length records" over a single DASD. In the other case, the elaboration defines variable length records in row major order (DASD track direction) over an array of DASDs. This substantially decreases data throughput needed for intensive computing in favor of increased concurrency (at least two different processes may access the array at the same time). It should also be noted, that execution of an update write command requires at least two DASD disk revolutions to completion.

This invention seeks to provide a method and means for minimizing the time required to access variable length records defined over an array of N+2 synchronized fixed block formatted DASDs and to make an efficient use of storage space thereon.

The invention also seeks to ensure that the access time to any array record location be no more than that required for a single pass even in the presence of a single DASD failure other than where the DASD containing the start of record has failed.

The invention further seeks to ensure that the access time to any array location be no more than that required for two passes where the DASD containing the start of record has failed.

Yet further, the invention seeks to enhance DASD array use as a fast access. high capacity, intermediate result storage for numerically intensive computations and the like.

The invention is based on the unexpected observation that a single pass access to any record on a synchronous DASD array can be ensured by:

(1) partitioning records into a string of equal sized blocks and writing the blocks in column major order across the DASDs,

(2) recording each start of record (count field) on different track extents of the same DASD in the array, and

(3) forming a parity block spanning one block recorded on the prior adjacent track extent on the DASD containing start of record blocks plus N blocks from the current track extent (column) of the N other DASDs.

Preferably, each variable length record is partitioned into a variable number K of equal fixed length blocks, and the blocks are written simultaneously in column major order onto the track extents of (N+1) DASDs. The column major order is constrained so that the first block of each record is written along a different track extent on the same track on the (N+1)th DASD.

Advantageously, a parity block P(i) is formed and written along an ith track extent on an (N+2)nd DASD corresponding to each ith track extent on the (N+1) DASDs. P(i) logically combines the block written along the (i-1)th track extent of the (N+1)st DASD and N blocks from the first N other DASDs along their ith track extent.

Conveniently, in response to each external (READ/WRITE) command, the tracks of the array DASDs are traversed in the order defined by the above mentioned steps, whereby the blocks forming any record specified in such command and the spanning parity blocks are accessed during a single pass.

In a method according to this invention, the column major order is taken K modulo (N+1). Also, the K blocks of any record occupy no more than K/(N+1) contiguous track extents on any one of the (N+1) DASDs. Consecutive extents along the same DASD track are also referred to as the row major order in an array. Lastly, the parity image is formed by exclusive OR (XOR) operation over the designated blocks in the same and offset columns.

The offset reflects the fact that the contents of the block appearing in the current track extent of the (N+1)th DASD are not always determinable by the system for a write operation. In contrast, such determination is always possible for the blocks appearing in the current track extent (column) of DASDs 1 to N.

Single pass access to variable length records, even where the array is subject to an opportunistic failure of a non-start of record containing DASD, is attained by concurrently XORing (reconstituting) and accessing the blocks on-the-fly using the the blocks from N+1 remaining DASDs.

Upon failure of the DASD containing the start of record, then two pass record access can be achieved for write access, while only one pass is needed for a read access. With respect to execution of a write command, the first pass is required to determine the start of record blocks while the second pass executes the access itself.

Unless and until the data from the failed DASD is rewritten onto a formatted spare DASD, the array is said to be operating in degraded mode. When the array operates with an updated spare it is said to be operating in fault tolerant mode. In the absence of spareing, the unavailable data must be recalculated for each access. Also, any additional DASD failure renders the array inoperable.

Consecutive byte ordering within each block, consecutive block ordering within each field, and consecutive field ordering within each record, are achieved.

The scope of the invention is defined in the appended claims; and how it can be carried into effect is hereinafter particularly described with reference to the accompanying drawings in which :-

Figure 1 depicts a synchronous array of (N+2) DASDs attached to a CPU by way of an array control unit;

Figure 2 illustrates a layout of variable length CKD records in row major order (track direction) across an N DASD array;

Figure 3 shows part of the controller logic of the array control unit of Figure 1; and

Figure 4 shows the layout of variable length CKD records onto an array of fixed block formatted DASDs in column major order (vertical track extent direction) according to the invention.

In this specification, "single pass" means an interval defined by a single period of rotation of a DASD disk, "two-pass" means an interval defined by two periods of rotation of a DASD disk.

A CPU 1 (Figure 1) accesses DASDs 1 to N+2 over a path in which an array control unit 2 includes channel. 3, array controller 5 and cache 13. Controller 5 operatively secures synchronism and accesses among DASD 1 to N+2 over address and control path 7. Responsive to an access, N+2 streams of data defining a predetermined number of consecutive bytes can be exchanged in parallel to cache. 13 over data path 15. Likewise, data can be exchanged serially by byte between CPU 1 and controller 5 over channel 3 after a parallel to serial conversion in controller 5 in the read direction and a serial to parallel conversion in the write direction.

In the read direction, data is supplied from cache 13 to controller 5 via data paths 9 and 11. In the write direction, data is moved from the controller 5 to the cache 3 over paths 9 and 11.

N+1 of the blocks represent at least a portion of a variable length record. The (N+2)th block contains parity information.

As a physical storage system, an array of N+2 DASDs is defined to be any physical arrangement of N+2 DASDs, selected ones (or all) of which can be accessed concurrently. Relatedly, the formatting and subsequent read/write accessing of an array, as a logical/physical store proceeds by copying/inserting values in consecutive positions on either a row or a column basis. If the operation is performed in a column direction, it is designated as being performed in "column major order". Likewise, if performed a row direction, it is designated as being performed in "row major order". Next, the mapping is done from the logical "array" to the physical store (i.e.ganged group of DASDs).

In a row major or track oriented layout (Fig. 2) of CKD variable length records according to the copending application, a parity is written onto a dedicated one of the array DASDs.

CPU 1 (Fig. 1) may be of the IBM S/370 type having an IBM MVS operating system, whose principles of operation are fully described in US-A-3,400, 371. A configuration involving CPUs sharing access to external storage is disclosed in US-A-4,207,609.

US-A-4,207,609 and the references cited therein describe CKD commands and their use by a CPU in obtaining variable length records from an attached DASD storage subsystem.

Under this architecture, the CPU creates a dedicated virtual processor for accessing and transferring data streams over demand/response interfaces to attached subsystems using chains of special purpose I/O instructions termed "channel command words" or CCWs. The CCWs are stored in a portion of CPU main memory in support of fast calls. When an application program executes a read or write requiring access to external storage (usually attached DASD storage), then the CPU S/370 operating system initiates such a reference with a START I/O command. This command causes the CPU to suspend its multi-processing state, transfer to the CCW chain, and re-establish its prior state after CCW chain completion.

At least some of the CCWs are sent by the CPU to the external storage subsystem for local interpretation or execution. That is, such CCWs as SEEK and SET SECTOR require the local dispatch of the accessing means to synchronize the movement of data to and from DASDs and the other system components. However, each independent reference or invocation of a CCW chain requires invocation of another START I/O MVS instruction.

Disadvantageously, each START I/O decreases overall CPU throughput because of the overhead cycles expended to save and restore the CPU information state. In this regard, the CKD command set was augmented by several commands to permit the external storage subsystem such as a DASD array to execute a series of operations without requiring a new START I/O.

ECKD is an acronym for Extended Count, Key, Data commands, and "IBM 3990 Storage Control Reference", 2nd edition, Copyright IBM 1988, GA32-0099, chapter 4 entitled "Command Descriptions" between pages 49 and 166 gives detailed architectural descriptions of the DEFINE EXTENT, LOCATE, READ, and WRITE commands. These commands are used in the subsequent description of the method and means of this invention as applied to CKD formatted variable length records.

CPU 1 issues DEFINE EXTENT and LOCATE as a pair of sequential commands to array controller 5. The first or DEFINE EXTENT command defines the boundaries or extent of array storage space that subsequent CCWs can access. The second or LOCATE command identifies the operation to be carried out on the data within the permitted space specified by the first command. Additionally, the LOCATE CCW points to the location

within the permitted space for executing the operation.

Array controller 5 stores the extent information in the first CCW and compares the extent information with the location mentioned in the second CD. The operation named by the. LOCATE CCW is executed upon a comparison match. Otherwise, controller 5 provides an error indication to CPU 1. Thus, multiple operations are permitted without incurring another START I/O.

In an embodiment of the invention (Fig. 4) involving the storage of variable length CKD formatted records, each one has a variable number K of equal sized fixed length blocks, in column major order (vertically) across the DASD array. Each fixed length block typically consists of 512 bytes. In this figure, one track from each of the four DASDs is set out. Illustratively, each DASD track capacity is up to 6 blocks.

The data blocks are consecutively numbered 1 to 18 in column major order. Likewise, the parity blocks are designated from P1 to P6. To facilitate discussion, it is assumed that the CKD tracks have no key fields. Each track containing CKD records comprises the traditional home address (HA) and and record 0 (RO) fields, followed by four records R1, R2, R3 and R4.

Each record has a count field and a data field. C1 is the count field for R1, C2 for R2, and so on. D1 is the data field for R1, D2 for R2, and so on. It is assumed that D1 is 1 kbyte, D2 is 512 blocks, D3 is 2.5 kblocks and D4 is 1 kblock.

The constraints are set forth as rules or numbered statements as follows:

1. Number all fixed length block in column major order. Thus, block 1 is on DASD 1, block 2 is on DASD 2, and so on, with block 4 back on DASD 1. Generalized, this states that a variable length record formed from K fixed blocks can be written in column major order K modulo (N+1).
2. Each block can contain data from only one CKD field
3. All fields are stored block-interleaved, not byte-interleaved. For example, the first 512 blocks of D1 are stored in block 4, and the second 512 blocks of D1 are stored in block 5.
4. All count fields must be stored in blocks resident on a designated other (N+1)th DASD (DASD 3 in Fig.4). Thus, C1 is in block 3 (from DASD 3), C2 is in block 6 (from DASD 3) and C3 is in block 9 (from DASD 3). D2 ends in block 7 and block 8 is not used in order to start C3 in block 9 on DASD 3.
5. Offset the column parity P(i) span to include one block from one track extent position (column i-1) earlier on the (N+1)th DASD plus N blocks from the current track extent position (column i) of the first N DASDs. P2 contains the parity from blocks 3, 4 and 5, P3 contains the parity from blocks 6, 7 and 8, and so on. Blocks 3, 4 and 5 belong to one parity group, blocks 6, 7 and 8 belong to another parity group, and so on.

In the absence of rule 5, P1 would contain the parity from blocks 1, 2 and 3, P2 would contain the parity from blocks 4, 5 and 6, P3 would contain the parity from blocks 7, 8 and 9, and so on.

Time units corresponding to track extents along a DASD track extent are denominated T1, T2, ..., T6 (Fig. 4). Upon a request to update D1, which would first require a search on C1, controller 5 conducts such a search in time unit T1 by reading C1 from block 3 into a buffer (not shown) in ECKD command interpreter and address logic 501 (Fig. 3) and compares its value against a host supplied value. On a match, the logic 501 issues commands to cache 13 over path 515, buffer and striping logic 503, and path 509 and to DASDs 1 and 2 over path 7 to write updated D1 in time unit T2 into blocks 4 and 5. At the same time, logic 501 also issues a command to cache 13 over path 513, offset parity coder 507, and to DASD 4 also over path 7 to write parity in block P2.

The parity in P2 consists of the exclusive OR of blocks 3, 4 and 5. Block 3, containing C1, was read at time T1, and blocks 4 and 5 contain the new values to be written in time T2, and are therefore also available in time. Therefore, all values to be XORed are available, and the parity can be generated in time and written into P2 in time T2. A read from DASD 3 is made in T1, and writes to DASDs 1,2 and 4 in time T2.

If there is a write to D2 followed by a write to D3, the operation proceeds as follows. In time T2, C2 is read from block 6 on DASD 3. On a match of C2, D2 is written on DASD 1 and P3 on DASD4, and C3 is read from DASD3, all in time unit T3. On a match of C3, in time T4, D3 is written to DASDs 1, 2 and 3 (blocks 10, 11 and 12), and P4 is written to DASD 4. Finally, in time T5, D3 is written to DASDs 1 and 2 (blocks 13 and 14), and P5 is written to DASD 4.

One pass writes are still possible even if either DASD 1 or DASD 2 has failed. A second pass will be needed if DASD 3 (which contains the count fields) has failed. Read operations always take one pass - with or without DASD failures. During such read operations, blocks read from DASD 3 will have to be delayed by one block time before being passed to the parity unit, so that blocks from DASDs 1 and 2 that belong to the same parity group may all arrive at the parity unit at the same time as the delayed block from DASD 3.

The space utilization of this method depends on the record lengths of CKD records. Excellent space utilization is possible with standard record lengths like 4k blocks which occur often in practice. With 4k records, the array uses 9 blocks (4.5 Kblocks) to store a CKD record, while a CKD DASD would use 4k blocks plus 40 bytes for the count field.

There are several ways to replace a failed DASD. These include manual or automatic substitution of the

failed DASD by a formatted DASD spare and the rebuilding of the data on the spare. US-A-4,914,656, and copending European patent application No. 92300586.2 describe so called "hot spareing" in RAID type 3 and RAID type 5 arrays, respectively.

Typically, array controller 5 would ascertain that one of the DASDs, say DASD1, has failed. The controller would switchably interconnect the spare in the manner described in US-A-4,914,656 and then systematically regenerate the data on the spare by XORing the contents of the remaining array DASDs in a single pass.

The key to restoring the array to fault tolerant mode is by writing the spare DASD other than the (N+1)th or (N+2)th with a one block offset and resynching the indices. The protocol for effectuating this is illustrated in the pseudo-code control flow shown below in Example 5.

Channel programs are sequences or chains of CCWs some of whose instructions are sent by CPU 1 for execution (technically interpretation) to array controller 5. The CCW chains are designed to simply read or write the "Channel". At the highest level, each CCW sequence consists of the commands DEFINE EXTENT, LOCATE, READ/WRITE, and test for terminating condition otherwise LOOP.

Five sequences are given as examples hereinafter. These include read and write channel programs for fault tolerant operation of the array, read and write channel programs for degraded mode array operation, and a redo of data from a failed DASD onto a substitute or spare. Each program is described in terms of the array controller and DASD dynamics.

In the following examples, a 3+P array is assumed. That is, three DASDs are dedicated to storing data and 1 DASD dedicated to storing parity blocks. One of the DASDs, DASD3 (Fig. 4) also stores start of record (count fields).

EXAMPLE 1

- START OF READ CHANNEL PROGRAM - FAULT TOLERANT MODE

Define Extent

Locate

Read

1. On receiving Define Extent:

   - Validate the command
   - Fetch and verify the parameters
   - Save the extent information (range of addresses)

2. On receiving Locate .

   - Validate Command
   - Fetch Cylinder and Head to seek to (CC, HH) respectively.
   - Validate CC, HH and ensure they are within extent specified before.
   - Fetch 5 bytes of search parameters. (CC,HH,R)
   - Validate search parameters
   - Fetch track extent number (S) parameter
   - Validate track extent number
   - Round S to nearest multiple of 3 (for 3+P array); say S'
   - Let S"=S'/3
   - Move all DASDs in array to cyl CC, Head HH, track extent S"

(In Figure 4 example, if S was specified as 8, This is rounded up to 9, then divided by 3 to get S" as 3. All DASDs are moved to track extent position 3, giving D2 on DASD 1, zeros on DASD 2 and C3 on DASD 3).

R: Read count field at this position, track extent S" from DASD 3. Other DASDs do nothing at this track extent position.

- If this is not a count field on DASD 3, increment S" by 1 and in next time unit, repeat the operation of the previous step. If it is a count field, continue to next step.
- Compare count field with search parameters (CC,HH,R) fetched earlier. If not equal, use key and data length parameters in count field to figure out track extent at which next count field will be. Update S" to this track extent position and go back to R:.

If search parameters match, fetch next CCW (Read Data)

- Use data length parameter in count field to figure out X, the number of track extents in the data field.

LOOP:

- If X>= 3, then read from next track extent position on all data DASDs
- If X = 2, then read from next track extent position on DASDs 1 and 2
- If X = 1, then read from next track extent position on DASD 1.
- If X < = 3, then stop. else X=X-3; and return to LOOP:
- END OF READ CHANNEL PROGRAM - FAULT TOLERANT MODE

****

EXAMPLE 2: - START OF WRITE CHANNEL PROGRAM - FAULT TOLERANT MODE

Define Extent

Locate

Write 1. On receiving Define Extent:

- Validate the command
- Fetch and verify the parameters
- Save the extent information

2. On receiving Locate

- Validate Command
- Fetch Cylinder and Head to seek to (CC, HH) respectively.
- Validate CC, HH and ensure they are within extent specified before.
- Fetch 5 blocks of search parameters.
- Validate search parameters
- Fetch track extent number (S) parameter
- Validate track extent number
- Round S to nearest multiple of 3 (for 3+P array); say S'
- Let S"=S'/3
- Move all DASDs in array to cyl CC, Head HH, track extent S" (In Figure 4 example, if S was specified as 8, This is rounded up to 9, then divided by 3 to get S" as 3. All DASDs are moved to track extent position 3, giving D2 on DASD 1, zeros on DASD 2 and C3 on DASD 3).

R: Read count field at this position, track extent S" from DASD

3. Other DASDs do nothing at this track extent position.

- If this is not a count field on DASD 3, increment S" by 1 and in next time unit, repeat the operation of the previous step. If it is a count field, continue to next step.
- Compare count field with search parameters fetched earlier. If not equal, use key and data length parameters in count field to figure out track extent at which next count field will be. Update S" to this track extent position and go back to R:.

If search parameters match, fetch next CCW (Write Data)

- Use data length parameter in count field to figure out X, the number of track extents in the data field.
- Curr_track extent=S"

LOOP:

- Prev_track extent = S"
- Curr_track extent = Curr_track extent+1
- If X >= 3, then write to curr_track extent position on all data DASDs
- If X = 2, then write to curr_track extent position on DASDs 1 and 2
- If X = 1, then write to curr_track extent position on DASD 1, and zeros to DASD 2.
- In all cases, write P to parity DASD at curr_track extent, where P is

  (prev_track extent on DASD 3) XOR (curr_track extent on DASD 1) XOR (curr_track extent on DASD 2)

- If X <= 3, then stop. else X=X-3; and return to LOOP:

- END OF WRITE CHANNEL PROGRAM - FAULT TOLERANT MODE.

Channel Program Execution in Degraded Mode:

Assume DASD 1 has failed

****

EXAMPLE 3

- START OF READ CHANNEL PROGRAM - DEGRADED MODE

Define Extent

Locate

Read

1. On receiving Define Extent:

- Validate the command

- Fetch and verify the parameters

- Save the extent information

2. On receiving Locate

- Validate Command

- Fetch Cylinder and Head to seek to (CC, HH) respectively.

- Validate CC, HH and ensure they are within extent specified before.

- Fetch 5 blocks of search parameters.

- Validate search parameters

- Fetch track extent number (S) parameter

- Validate track extent number

- Round S to nearest multiple of 3 (for 3+P array); say S'

- Let S"=S'/3

- Move all DASDs (except failed DASD) in array to cyl CC, Head HH,

track extent S".

(In Figure 4 example, if S was specified as 8, this is rounded up to 9, then divided by 3 to get S" as 3. All DASDs are moved to track extent position 3, giving D2 on DASD 1, zeros on DASD 2, C3 on DASD 3, and P3 on DASD 4 (the parity DASD)).

R: Read count field at this position, track extent S" from DASD 3. Other DASDs do nothing at this track extent position.

- If this is not a count field on DASD 3, increment S" by 1 and in next time unit, repeat the operation of the previous step. If it is a count field, continue to next step.
- Compare count field with search parameters fetched earlier. If not equal, use key and data length parameters in count field to figure out track extent at which next count field will be. Update S" to this track extent position and go back to R:

If search parameters match, fetch next CCW (Read Data)

- Use data length parameter in count field to figure out X, the number of track extents in the data field.
- Curr_track extent=S"

LOOP:

- Prev_track extent = S"
- Curr_track extent = Curr_track extent+1
- If X>= 3, then read from curr_track extent on all DASDs. Since DASD 1 is broken, calculate curr_track extent on DASD 1 on-the-fly as

(prev_track extent on DASD 3) XOR (curr_track extent on DASD 2) XOR (curr_track extent on DASD 4)

prev_track extent on DASD 3 is now available, and curr_track extent on DASDs 2 and 4 can be read. Also, curr_track extent on DASD 3 should be read , since this will become prev_track extent on next iteration of loop, when prev_track extent on DASD 3 may be needed.

- if X = 2, then return curr_track extent from DASDs 1 and 2 to CPU 1 host as part of the read. Curr_track extent from DASD 2 is directly read and returned to CPU.

Curr_track extent from DASD 1 is calculated on-the-fly as

(prev_track.extent on dev. 3) XOR (curr_track extent on dev. 2) XOR (curr_track extent on DASD 4)

prev_track extent already located on DASD 3, and curr_track extent can be read from DASDs 2 and 4. Also, curr_track extent should also be read from DASD 3, since this will become prev_track extent on next iteration of LOOP, when prev_track extent on DASD 3 may be needed.

- if X = 1, then it is necessary to return curr_sector from DASD 1 to CPU 1. Since DASD 1 is broken, curr_track extent from DASD 1 is calculated on-the-fly as

(prev_track extent on DASD 3) XOR (curr_track extent on DASD 2) XOR (curr_track extent on DASD 4)

prev_track extent on DASD 3 is already available, and curr_track extent can be read from on DASDs 2 and 4. curr_track extent should also be read fromn DASD 3, since

this will become prev_track extent on next iteration of loop, when prev_track extent on DASD 3 will be needed.

- If X<=3, then stop. else X=X-3; and return to LOOP:
- END OF READ CHANNEL PROGRAM - DEGRADED MODE

****

EXAMPLE 4:

- START OF WRITE CHANNEL PROGRAM - DEGRADED MODE

Define Extent

Locate

Write

1. On receiving Define Extent:

- Validate the command
- Fetch and verify the parameters
- Save the extent information

2. On receiving Locate

- Validate Command
- Fetch Cylinder and Head to seek to (CC, HH) respectively.
- Validate CC, HH and ensure they are within extent specified before.
- Fetch 5 bytes of search parameters.
- Validate search parameters
- Fetch track extent number (S) parameter
- Validate track extent number.
- Round S to nearest multiple of 3 (for 3+P array); say S'
- Let S"=S'/3
- Move all DASDs in array (except broken DASD)

to cyl CC, Head HH, track extent S"

(In Figure 4 example, if S was specified as 8, this is rounded up to 9, then divided by 3 to get S" as 3. All DASDs are moved to track extent position 3, giving D2 on DASD 1, zeros on DASD 2, C3 on DASD 3 and P3 on DASD 4 (the parity DASD)).

R: Read count field at this position, track extent S" from DASD 3. Other DASDs do nothing at this track extent position.

- If this is not a count field on DASD 3, increment S" by 1 and in next time unit, repeat the operation of the previous step. If it is a count field, continue to next step.
- Compare count field with search parameters fetched earlier. If not equal, use key and data length parameters in count field to figure out track extent at which next count field will be. Update S" to this track extent position and go back to R:.

If search parameters match, fetch next CCW (Write Data) - Use data length parameter in count field to figure out X, the number of track extents in the data field.

- Curr_track extent=S"

LOOP:

- Prev_track extent = S"
- Curr_track extent = Curr_track extent+1
- If X>= 3, then write to curr_track extent position on all DASDs except DASD 1
- if X = 2, then write to curr_track extent position on DASDs 2

and 4, and read from curr_track extent on DASD 3. Curr_track extent on DASD 3 is read, because, in the next iteration, it will become prev_track extent, and prev_track extent on DASD 3 is needed to generate parity.

- if X = 1, then write to DASD 2 (zeroes) and DASD 4. Read from curr_track extent on DASD 3. Curr_track extent on DASD 3 is read, because, in the next iteration, it will become prev_track extent, and prev_track extent on DASD 3 is needed to generate parity.
- In all cases, write P to parity DASD at curr_track extent, where P is

  (prev_track extent on DASD 3) XOR

  (curr_track extent on DASD 1)

  XOR (curr_track extent on DASD 2) Curr_track extent on DASD 1 in above formula is obtained from the host as part of the Write data and is not read from DASD. Prev_track extent on DASD 3 was read at the previous sector position.
- If X<=3, then stop. else X=X-3; and return to LOOP:
- END OF WRITE CHANNEL PROGRAM - DEGRADED MODE
- At the end of write, no data has been written to DASD 1, but data has been written to all other data DASDs and the parity DASD, so that when the failed DASD is replaced, all the missing data can be recalculated and stored on the replaced DASD.

****

EXAMPLE: 5

METHOD FOR REBUILDING DATA AFTER FAILED DASD 1 HAS BEEN

REPLACED WITH A FORMATTED SPARE DASD:

- C = # of cylinders on DASD
- T = # of tracks per cylinder
- B = # of track extents per track
- Attach formatted spare DASD to array controller and align the track indices offset by one track extent from the indices of the other array DASDs

```
Do from i = 1 to C;
 Move to cylinder i on all DASDs
   Do from j = 1 to T
     Move to track j on all DASDS
       Do from k = 1 to B
         Generate track extent k to be stored on spare DASD in
         time unit k
         value of track extent k on DASD 1 (spare DASD) is
         generated as
            (track extent k on DASD 2) XOR
            (track extent k on DASD 4) XOR
            (track extent k-1 on DASD 3)
         when k>=2;
         value of track extent k on DASD 1 is generated as
            (track extent k on DASD 2) XOR
             (track extent k on DASD 4) XOR
            (track extent B on DASD 3)
         when k=1;
         store the generated track extent k on the spare DASD
```

```
            at time unit k+1
          end do on k;
        end do on j;
      end do on i;
      resynchronise track indices of spare DASD with the track
      indices of the remaining array DASDs
      ENTER FAULT TOLERANT MODE
*****************************************************************
```

Claims

1. A method of accessing variable length records to and from a fixed block formatted synchronous DASD array of N+2 DASDs, comprising the steps of :-
   (1) partitioning records into strings of equal sized blocks in column major order across the DASDs,
   (2) recording each start of record (count field) on different track extents of the same DASD in the array, and
   (3) forming a parity block spanning one block recorded on the prior adjacent track extent on the DASD containing start of record blocks plus N blocks from the current track extent (column) of the N other DASDs.

2. A method according to claim 1, characterised in that in step (1) each variable length record is partitioned into a variable number K of fixed length blocks, and the blocks are written synchronously in column major order onto the track extents of (N+1) DASDs, the column major order being constrained so that the first block of each record is written along a different track extent on the same track on the (N+1)th DASD.

3. A method according to claim 1 or 2, characterised in that in step (3), concurrently with step (1), a parity block P(i) is formed and written along an ith track extent on an (N+2)th DASD corresponding to an ith track extent on the (N+1) DASDs, P(i) logically combining the block written along the (i-1)th track extent of the (N+1)th DASD and N blocks from the first N other DASDs along their ith track extent.

4. A method according to claim 3, as appendant to claim 2, characterised in that, in response to each external (READ/WRITE) command, the tracks of the array DASDs are traversed in the order defined, whereby the blocks forming any record specified in such command and the spanning parity blocks are accessed during a single pass.

5. A method according to any preceding claim, wherein the column major order is taken K modulo (N+1), and the K blocks of any record occupy no more than K/(N+1) contiguous track extents on any one of the first (N+1) DASDs.

6. A method according to claim 3, or any claim appendant to claim 3, wherein the logical combining includes that of performing an exclusive OR operation over N+1 blocks.

7. A method of single pass synchronous access to variable length records across ones of a fixed block formatted DASD array of N+2 DASDs over a common path, each record including at least a fixed length count field and a variable length data field, comprising the steps of:
   (a) reformatting each record into a first block representing the count field and a variable number of other blocks representing the data field, and, synchronously writing the blocks of each reformatted record in column major order over (N+1) DASDs, the column major order being constrained such that all first blocks are recorded on the (N+1)th DASD;

(b) forming and writing a parity block for each counterpart column order concurrent with step (a) on the (N+2)th DASD, each parity block spanning N blocks in the same column from the first N DASDs and one block one column offset thereto on the (N+1)th DASD; and
(c) responsive to each command requesting at least one record, accessing the blocks forming the requested record and the correlated parity blocks in the order defined by steps (a) and (b), thereby ensuring access of the requested record and its parity image in a single pass.

8. A method according to claim 7, wherein the parity block is formed by XORing of the spanned blocks.

9. A method according to claim 7, wherein each counterpart track on each array DASD includes an indexed reference point, step (c) of the method being modified responsive to an opportunistic DASD failure other than the (N+1)th and (N+2)th DASDs, by recovering each unavailable block from the correlated parity block and the remaining blocks spanned by the parity block and concurrently accessing the blocks of the requested record including any recovered blocks in the column major order defined by steps (a) and (b).

10. A method according to claim 9, in which step (c) is modified responsive to an opportunistic failure of the (N+1)th DASD and a command requesting at least one record, the command being selected from the set consisting of the read, write, and write update commands, to include steps of recovering the first blocks of each record from the correlated parity block and the remaining blocks spanned by the parity block during a first pass, and during execution of any write update command, accessing the blocks of the requested record including any recovered blocks in the column major order defined by steps (a) and (b) during a second pass.

11. A method according to claims 9 or 10, wherein the recovering of unavailable blocks includes the XORing of the correlated parity block and the remaining blocks spanned by the parity block.

12. A method according to claim 9, 10 or 11, wherein at least one of the DASDs of the array is reserved, whereupon, if any of the DASDs other than the spare, the (N+1)th, and the (N+2)th is rendered unavailable for record accessing, the method includes the steps of replacing the unavailable DASD by the reserved DASD, and, in a single pass, recovering each unavailable block by XORing each correlated parity block and the remaining available blocks spanned by the parity block, and writing each recovered block to the spare DASD with a one block offset, and resynching the indices of the array DASDs.

13. An array control unit (ACU) responsive to external commands for synchronously accessing DASDs in N+2 fixed block formatted cyclic multitracked storage devices (DASDs), and for transferring one or more variable length records between DASDs and a CPU coupled thereto, wherein the array control unit comprises:
(a) means (3,501,503) responsive to an external write command for partitioning a variable length record into variable number K of consecutive fixed length blocks;
(b) means (7,13,15) for writing the blocks in column major order over (N+1) DASDs in which the first block of each record is written on the (N+1)th DASD;
(c) means (507,511,513) for forming and writing a parity block for each counterpart column order concurrent with step (a) on the (N+2)th DASD, each parity block spanning N blocks in the same column from the first N DASDs arid one block one column offset thereto on the (N+1)st DASD; and
(d) means (501,7,503,507) responsive to each external command requesting at least one record, accessing the blocks forming the requested record and the correlated parity blocks in the order defined by means (a), (b) and (c), thereby ensuring access of the requested record and its parity image in a single pass.

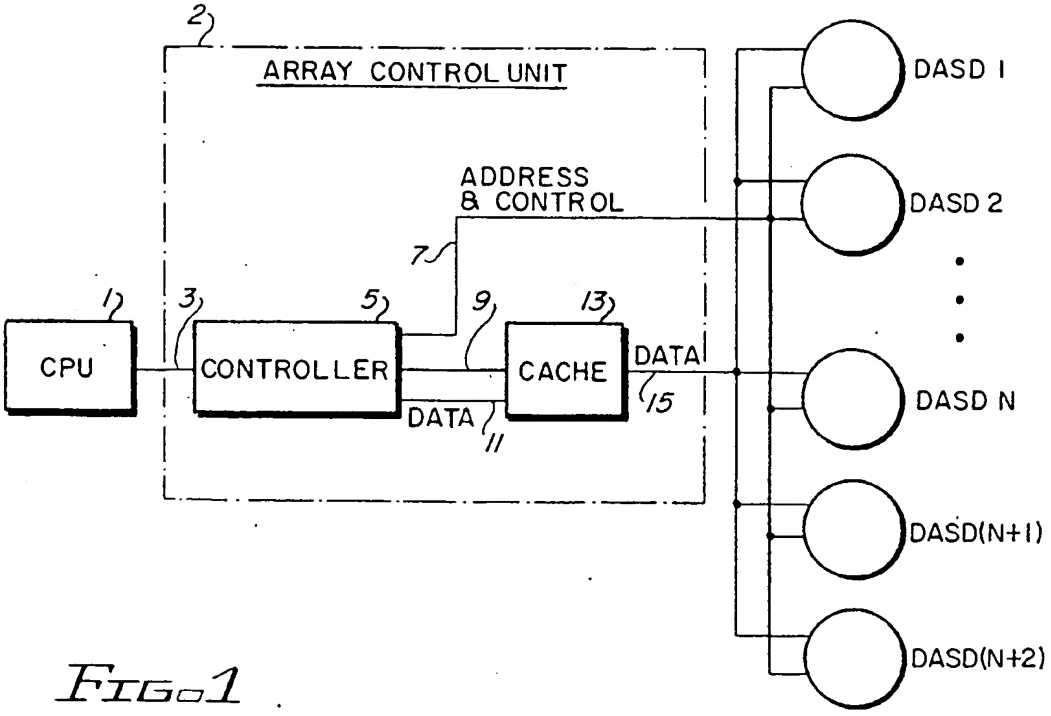
2
ARRAY CONTROL UNIT
ADDRESS
& CONTROL
7
1
3
5
9
13
CPU
CONTROLLER
CACHE
DATA
15
DATA
11
DASD 1
DASD 2
DASD N
DASD(N+1)
DASD(N+2)

FIG. 1

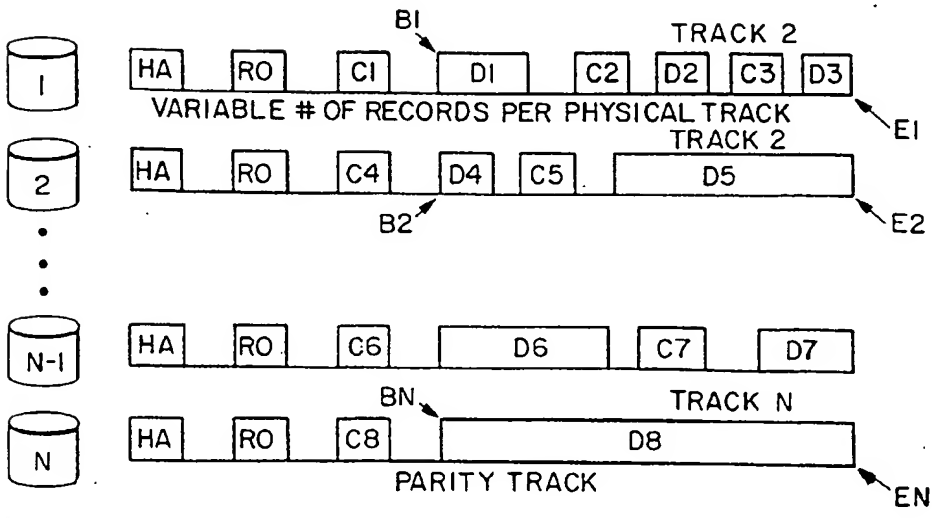
BI
TRACK 2
HA
RO
CI
DI
C2
D2
C3
D3
1
VARIABLE # OF RECORDS PER PHYSICAL TRACK
EI
TRACK 2
HA
RO
C4
D4
C5
D5
2
B2
E2
N-1
HA
RO
C6
D6
C7
D7
BN
TRACK N
HA
RO
C8
D8
N
PARITY TRACK
EN

FIG. 2

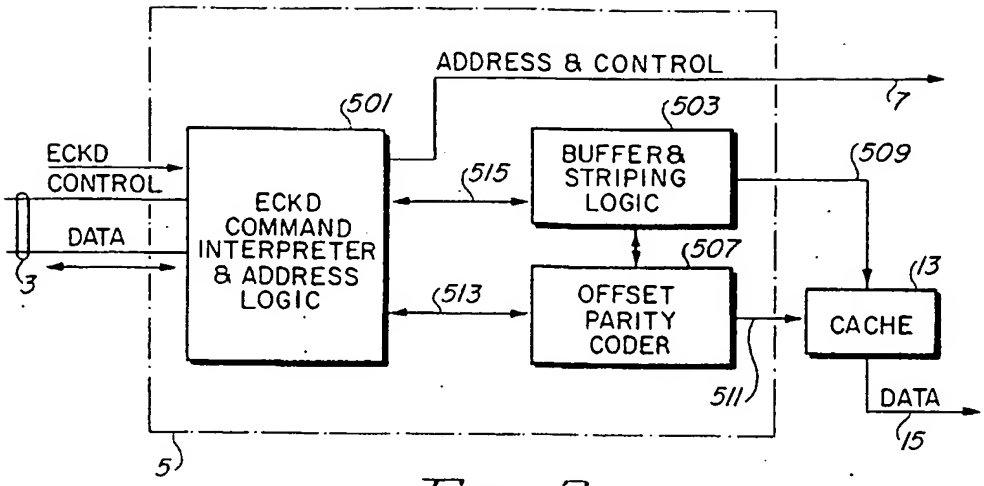
ADDRESS & CONTROL
7
501
503
ECKD
CONTROL
DATA
3
ECKD
COMMAND
INTERPRETER
& ADDRESS
LOGIC
515
BUFFER &
STRIPING
LOGIC
509
507
513
OFFSET
PARITY
CODER
13
CACHE
511
DATA
15
5

FIG. 3

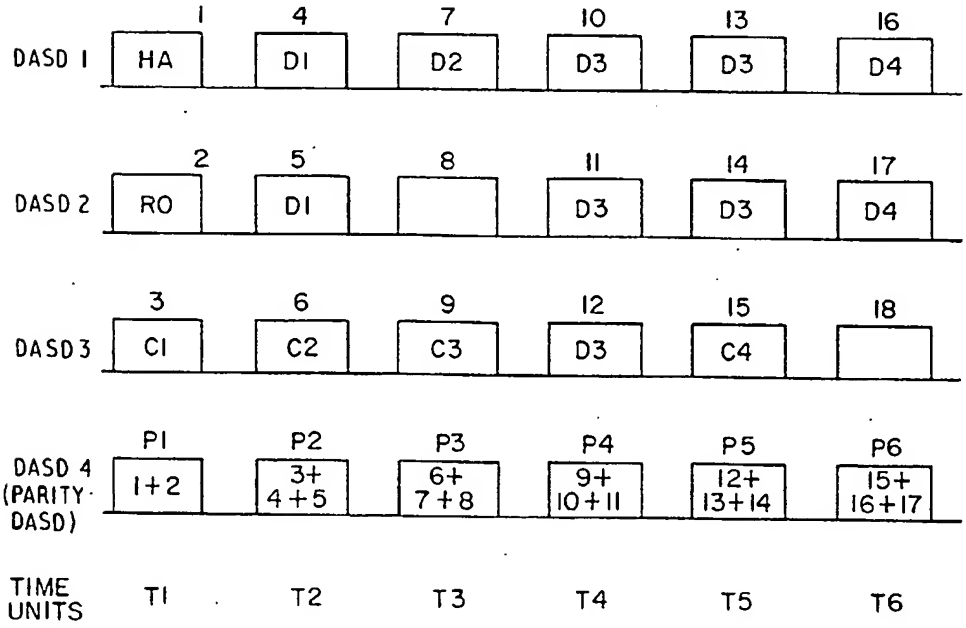
DASD 1
1
HA
4
D1
7
D2
10
D3
13
D3
16
D4
DASD 2
2
R0
5
D1
8
11
D3
14
D3
17
D4
DASD 3
3
C1
6
C2
9
C3
12
D3
15
C4
18
DASD 4
(PARITY
DASD)
P1
1+2
P2
3+
4+5
P3
6+
7+8
P4
9+
10+11
P5
12+
13+14
P6
15+
16+17
TIME
UNITS
T1
T2
T3
T4
T5
T6

FIG. 4